Department of Computer Science

University of Engineering and Technology, Lahore

Write the C++ Functions of the following problems.

Problem 1:

Create a function that counts the number of times a particular letter shows up in the word search. Users can enter any number of letters in the character array. First ask the user how many times he/she wants to enter elements in the character array.

Test Cases:

- 6
  ['D', 'E', 'Y', 'H', 'A', 'D']
  letterCounter('D') ⟶ 2 // "D" shows up 2 times

- 8
  ['D', 'E', 'E', 'H', 'E', 'D', 'T', 'B']
  letterCounter('E') ⟶ 3

Problem 2:

Write a function that stores all the elements in an array that are strictly greater than their adjacent left and right neighbors in a new array and then display that array.

Note:

- Do not count boundary numbers, since they only have **one** left/right neighbor.
- If no such numbers exist, return an empty array.

Test Cases:

- **How many Elements you want to Enter: 8**
  [4, 5, 2, 1, 4, 9, 7, 2]
  miniPeaks() ⟶ [5, 9] // 5 has neighbours 4 and 2, both are less than 5.

- **How many Elements you want to Enter: 9**
  [1, 2, 1, 1, 3, 2, 5, 4, 4]
  miniPeaks() ⟶ [2, 3, 5]

- **How many Elements you want to Enter: 6**
  [1, 2, 3, 4, 5, 6]
  miniPeaks() ⟶ []

Problem 3:
Create a main function that takes a number from the user and then an array of the size mentioned by the user and finds the integer which appears an odd number of times.
There will always only be one integer that appears an odd number of times.

Test Cases:

- Enter Length of Array: 11
  [1, 1, 2, -2, 5, 2, 4, 4, -1, -2, 5]
  findOdd() ⟶ -1

- Enter Length of Array: 13
  [20, 1, 1, 2, 2, 3, 3, 5, 5, 4, 20, 4, 5]
  findOdd() ⟶ 5

- Enter Length of Array: 1
  [10]
  findOdd() ⟶ 10

Problem 4:
Below is an example of a repeating cycle.

```
[1, 2, 3, 1, 2]
isRepeatingCycle(3) => true
// Since the first two elements of [1, 2, 3] equals [1, 2]
```

Below is an example of a **non-repeating cycle**.

```
[1, 2, 3, 1, 3]
isRepeatingCycle(3) => false
// Since [1, 2, 3] != [1, 3]
```

Take input from the user in a globally declared array and the length of cycle in local variable. You are tasked with writing a function that takes in 1 input as parameter:

The length of each cycle.

Return the boolean value true if the array is a repeating cycle, and false if the array is a non-repeating cycle. All cycles begin with the first element of the array. Return true if the cycle length is greater than the array length.

Test Cases:

- **Array Length: 7**
  [1, 2, 3, 1, 2, 3, 1]
  isRepeatingCycle(3) ⟶ true

- **Array Length: 7**
  [1, 2, 3, 4, 2, 3, 1]
  isRepeatingCycle(4) ⟶ false

- **Array Length: 5**
  [1, 2, 1, 2, 2]
  isRepeatingCycle(2) ⟶ false

- **Array Length: 4**
  [1, 1, 1, 1]
  isRepeatingCycle(3) ⟶ true

Problem 5:

Create a function that determines whether elements in an array can be re-arranged to form a consecutive list of numbers where each number appears exactly once.

Test Cases:

- Array Length: 5
  [5, 1, 4, 3, 2]
  cons() ⟶ true
  // Can be re-arranged to form [1, 2, 3, 4, 5]

- Array Length: 6
  [5, 1, 4, 3, 2, 8]
  cons() ⟶ false

- Array Length: 6
  [5, 6, 7, 8, 9, 9]
  cons() ⟶ false
  // 9 appears twice